Substituting Convolutions for Neural Network Compression

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 16, 2021, accepted May 20, 2021, date of publication June 4, 2021, date of current version June 16, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3086321

Substituting Convolutions for Neural Network Compression

ELLIOT J. CROWLEY[1,*], GAVIN GRAY[2,*], JACK TURNER[3], AND AMOS STORKEY[3]
[1]School of Engineering, The University of Edinburgh, Edinburgh EH8 9YL, U.K.
[2]Vector Institute, University of Toronto, Toronto, ON M5S 1A1, Canada
[3]School of Informatics, The University of Edinburgh, Edinburgh EH8 9YL, U.K.

Corresponding authors: Elliot J. Crowley (elliot.j.crowley@ed.ac.uk) and Gavin Gray (gray@vectorinstitute.ai)

This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) Scholarship from the Neuroinformatics and Computational Neuroscience Doctoral Training Centre, in part by the EPSRC Centre for Doctoral Training in Pervasive Parallelism, in part by the Huawei DDMPLab Innovation Research Grant, in part by the European Union's Horizon 2020 Research and Innovation Programme under Grant 732204 (Bonseyes), and in part by the Swiss State Secretariat for Education, Research and Innovation (SERI) under Contract 16.0159.

*Elliot J. Crowley and Gavin Gray contributed equally to this work.

ABSTRACT Many practitioners would like to deploy deep, convolutional neural networks in memory-limited scenarios, e.g. on an embedded device. However, with an abundance of compression techniques available it is not obvious how to proceed; many bring with them additional hyperparameter tuning, and are specific to particular network types. In this paper, we propose a simple compression technique that is general, easy to apply, and requires minimal tuning. Given a large, trained network, we propose (i) substituting its expensive convolutions with cheap alternatives, leaving the overall architecture unchanged; (ii) treating this new network as a student and training it with the original as a teacher through distillation. We demonstrate this approach separately for (i) networks predominantly consisting of full 3×3 convolutions and (ii) 1×1 or *pointwise* convolutions which together make up the vast majority of contemporary networks. We are able to leverage a number of methods that have been developed as efficient alternatives to fully-connected layers for pointwise substitution, allowing us provide Pareto-optimal benefits in efficiency/accuracy.

INDEX TERMS Machine learning, deep neural networks, computer vision, DNN compression.

I. INTRODUCTION

Deep neural networks [1] (DNNs) are able to excel at a multitude of challenging tasks in computer vision, such as image classification, object detection, and semantic segmentation of complex scenes. However, they are famously large and cumbersome, utilising millions (sometimes billions) of parameters, which makes deployment on edge devices very challenging. Arguably, deployment of DNNs to the edge is where they will see the greatest use [2], e.g. for pedestrian detection in a vehicle's computer or human activity recognition on wearables [3]. We want our networks to be smaller so they could, for instance, fit into the L1 cache of an ARM Cortex-A55 (32–128 KB). This is a challenge when many networks occupy several hundreds of MB.

So how do we achieve this? After all, we do not want to miss out on the expressive power of deep neural network architectures [4]–[10]. Fortunately, there is a wealth of literature on neural network compression. It is possible to prune a network to remove redundant connections [11]–[15] or quantise a network's weights or activations [16]–[20]. Unfortunately, many of these approaches rely on hyperparameter tuning and are architecture specific; pruning for example excels on networks that have large, redundant fully connected layers which are less prevalent in newer architectures [21].

In this paper, we propose a simple, and general compression technique, that relies on the observation that while architectures can differ, the vast majority of the parameter and computation budget in modern DNNs is used by convolutions: either **full convolutions** with 3×3 kernels, or increasingly, **pointwise convolutions** [7], [22], [23]; those with 1×1 kernels. Given a trained network, we construct a compressed *student* network by replacing its convolutions with a cheaper, substitute convolution, i.e. one utilising fewer parameters or operations; the structure of the network remains unchanged. We can then use the original network as

The associate editor coordinating the review of this manuscript and approving it for publication was Mostafa Rahimi Azghadi.

a *teacher* to train the student to high performance through distillation [24], [25].

For networks consisting predominantly of full convolutions, we use grouped convolutions [26] and bottlenecks [5] as our substitute operations. To construct a student network we replicate the teacher architecture while replacing each full convolution in the network with one of these substitute operations. We show that for a comparable number of parameters, these student networks outperform student networks with full convolutions and smaller architectures. This research formed part of a conference paper at NeurIPS [27].

We go further with networks utilising pointwise convolutions by remarking that a pointwise convolution is simply a fully-connected layer applied at each spatial location of an image representation. This allows us to leverage a number of methods that have been developed as efficient alternatives to fully-connected layers [28]–[30] that have seen little consideration in the convolutional setting. By using these methods to produce substitute operations for student networks, we can obtain high compression rates while retaining most of the original performance; exploring a new region in the Pareto Frontier in terms of accuracy/memory in the process.

This paper is structured as follows: In Section III we review a range of commonly-available substitute convolutions that may be used in-place of full convolutions to form a compressed student network. Then, we turn our attention to networks utilising pointwise convolutions: we detail how these can be substituted for a variety of efficient linear alternatives (Section IV). Finally in Section V we demonstrate that networks with such substitutions, trained through distillation, achieve high performance while being significantly smaller than their original counterparts.

To summarise, the contributions of the paper are:

- We present a simple compression technique where we replace a network's expensive convolutions with cheaper alternative convolutions. This is then treated as a student network, and is distilled from the original network which is used as a teacher.
- We propose cheap alternative convolutions for networks where the most expensive convolutions are (i) full 3×3 convolutions and ii) pointwise (1×1) convolutions.
- We perform comprehensive experiments showcasing the sucecess of our approach across CIFAR and ImageNet for different families of networks.

II. BACKGROUND

Our approach relies on replacing the expensive convolutions in a network and then distilling the knowledge from the original network into it. Many of our proposed replacements are based on grouped convolutions and efficient linear layers, which we review in Sections II-A and II-B respectively. We review distillation in Section II-C.

A. GROUPED CONVOLUTIONS

A grouped convolution is illustrated in Figure 1: the input tensor is split into groups of channels, independent filters

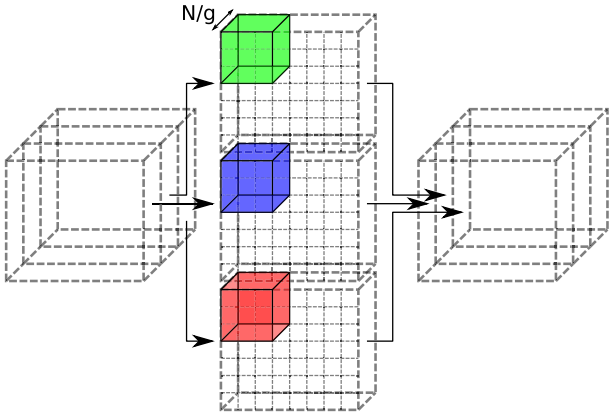

FIGURE 1. A grouped convolution operates by passing independent filters over the tensor after it is separated into g groups over the channel dimension; as each of the g filters needs only to operate over N/g channels, this reduces the parameter cost of the layer by a factor of g.

are passed over each of these groups, and the groups are then concatenated together again. Convolutions use parameters quadratically with the size of the channels. Performing independent convolutions over channel groups therefore uses fewer parameters. In the extreme, when there are as many channels as groups, the grouped convolution only uses parameters linear in the number of channels.

Grouped convolutions with two groups appeared in the original AlexNet paper [31] due to GPU memory constraints while training, but are now so prevalent that they are a part of cuDNN [32] which, along with CUDA, forms the most common backend for deep learning frameworks. We refer the reader to [33] for a recent, fast grouped convolution implementation for edge hardware.

Having the same number of groups as channels is the most limiting form of grouped convolution; information in any channel cannot influence another. To deal with this, a common solution is to use a pointwise convolution after the grouped convolution. The combination of the two is commonly called a separable convolution.

Sifre developed separable convolutions in their current form [26]. After this, they were used in the Xception architecture [34] for improved results in classification; at the same time speeding up inference at test time. The most significant application for efficiency has been their use in the MobileNet architecture [35] and in the Inception block [36]. MobileNet demonstrated separable convolutions for efficiency, achieving greater accuracy than SqueezeNet [37] while using 22 times fewer multiply-add (mult-add) operations. Using a number of groups not equal to the number of channels is less popular, but has been explored [38], [39].

Separable convolutions are similar to intra-channel convolutions [40]. However, in this case the *same filter* is applied to channels independently, and then channels are linearly combined, repeating the process for as many output channels as required. Despite making a parallel operation sequential, the technique has been demonstrated to make ResNets [5] use more than 4 times fewer mult-adds.

Another step to reduce parameter usage would be to group the pointwise convolution, but doing this would make channel groups disconnected throughout the network. To work around this problem, ShuffleNet [41] proposed a riffle shuffle of the channels in between alternating grouped pointwise convolutions. On mobile devices this network was more than twice as fast as MobileNet [35] within 3% error. CondenseNet [42] adopts a similar strategy.

Deconstructing convolutions further for efficiency has also been considered in the literature. 3D convolutions along the channels and spatial axes of the input tensor can be as effective as a traditional convolutional architecture [43]. Unlike most ways to modify the elements of neural networks, this approach shows that a network without the spatial kernel receptive field we have come to expect in deep learning can still classify natural images.

B. EFFICIENT LINEAR LAYERS

Efficient linear layers reduce the problem of efficient architecture design to that of rethinking the matrix multiply involved in a linear transformation. Convolution can also be implemented efficiently as a matrix multiplication [44], making these approaches general enough to consider when designing an efficient network.

Deep Fried Convnets [45] approximate a matrix multiplication using a Fastfood transform. The Fastfood transform is composed of permutations, Hadamard transforms and diagonal random matrices, which are the trainable parameters. The advantage of this sequence of transformations is that the number of operations scales log-linearly instead of quadratically. Following this, a similar method using the discrete cosine transform (DCT) was developed, named ACDC [30], which simply applies two diagonal matrices of parameters between a forward and reverse DCT. ACDC could operate twice as fast as a Deep Fried Convnet [45], speeding up an AlexNet [31] six times with only a 0.6% drop in accuracy.

Structured Spinners [46] have been proposed as a more general method based on a sequence of Hadamard and random diagonal matrices. This paper did not present results on large-scale image classification problems. As in all of these methods the particular parameterisation affects the convergence of stochastic gradient descent (SGD), which can be a barrier to adoption.

Despite the implementation of convolution in most frameworks as a matrix multiply, these techniques have seen little application to reduce the number of convolutional parameters in networks, although we refer the reader to [47] for a recent example.

C. DISTILLATION

An obvious way to produce a network for edge deployment would be to choose a much smaller convolutional network architecture to save computation and storage. Unfortunately this comes with a large hit in performance. However, if the smaller *student* model is trained on the outputs of the larger *teacher* model, it can perform close to the original on test data [24]. This process was popularised as knowledge distillation [25], and has also featured in reinforcement learning [48] and Bayesian scenarios [49].

We define knowledge distillation as follows: we denote the cross entropy of two probability vectors $\mathbf{p}$ and $\mathbf{q}$ as $\mathcal{L}_{CE}(\mathbf{p}, \mathbf{q}) = -\sum_k p_k \log q_k$. Assume we have a dataset of elements, with one such element denoted $\mathbf{x}$, where each element has a corresponding one-hot class label: denote the one-hot vector corresponding to $\mathbf{x}$ by $\mathbf{y}$. Given $\mathbf{x}$, we have a trained teacher network $\mathbf{t} = \text{teacher}(\mathbf{x})$ that outputs the corresponding logits, denoted by $\mathbf{t}$; likewise we have a student network that outputs logits $\mathbf{s} = \text{student}(\mathbf{x})$. To perform knowledge distillation we train the student network to minimise the following loss function (averaged across all data items):

$$\mathcal{L}_{KD} = (1-\alpha)\mathcal{L}_{CE}(\mathbf{y}, \sigma(\mathbf{s})) + T^2 \alpha \mathcal{L}_{CE}\left(\sigma\left(\frac{\mathbf{t}}{T}\right), \sigma\left(\frac{\mathbf{s}}{T}\right)\right),$$

where $\sigma(.)$ is the softmax function, T is a temperature parameter and α is a parameter controlling the ratio of the two terms. The first term is a standard cross entropy loss penalising the student network for incorrect classifications. The second term is minimised if the student network produces outputs similar to that of the teacher network.

Some theory has been developed to explain how this is possible, and is explored by [50] and [51]. The simplest explanation is that the information content of the logits of a trained network is much higher than the information content of a one-hot categorical vector, so it provides better supervision from which to learn. However, this is likely insufficient, because it is possible to train a group of student networks jointly, without the need for a teacher and obtain better results than if any of the networks had been trained individually [52]. The network morphisms of [53] can also be viewed as a kind of model distillation in which the added capacity is trained using the predictions of the smaller model, turning the distillation idea upside down.

Following the discovery of limitations in the original model distillation method [54], [55], interest in this method waned. Modern deep convolutional architectures could not be compressed into a less deep student architecture [54]. However, a method called *attention transfer* (AT) [56] using supervision at intermediate layers has demonstrated it is still possible to learn simpler residual networks [5] that can mimic larger ones. As we demonstrate in this work, we can use the attention transfer loss to distil a large network into a smaller network with minimal engineering effort by replacing the convolutions in the teacher with a cheap substitute.

We formally define attention transfer as follows: consider a choice of layers $i = 1, 2, \ldots, L$ in a teacher network with L layers, and the corresponding layers in the student network. At each chosen layer i of the teacher network, we will collect the spatial map of the activations for channel j into the vector $\mathbf{a}^t_{ij}$. We will let A^t_i collect $\mathbf{a}^t_{ij}$ for all j. Likewise for the student network we correspondingly collect into $\mathbf{a}^s_{ij}$ and A^s_i. Now given some choice of mapping $\mathbf{f}(A_i)$ that maps

TABLE 1. Convolutional blocks used as substitutions for full convolutional blocks: a standard block *S*, a grouped + pointwise block *G*, a bottleneck block *B*, and a bottleneck grouped + pointwise block *BG*. Conv refers to a $k \times k$ convolution. GConv is a grouped $k \times k$ convolution and Conv 1 × 1 is a pointwise convolution. Blocks use pre-activations [57]: all convolutions are preceded by a batch-norm layer + a ReLU activation. We assume that the input and output to each block has N channels and that channel size does not change over a particular convolution unless written out explicitly as ($x \rightarrow y$). Where applicable, g is the number of groups in a grouped convolution and b is the bottleneck contraction. We give the cost of the convolutions in each block in terms of these parameters. The batch-norm cost at test time is also given, but is markedly smaller.

Block	S	$G(g)$	$B(b)$	$BG(b,g)$
Structure	Conv	GConv (g)	Conv1x1($N \rightarrow \frac{N}{b}$)	Conv1x1($N \rightarrow \frac{N}{b}$)
	Conv	Conv1x1	Conv	GConv(g)
		GConv (g)	Conv1x1($\frac{N}{b} \rightarrow N$)	Conv1x1($\frac{N}{b} \rightarrow N$)
		Conv1x1		
Conv Params	$2N^2k^2$	$2N^2(\frac{k^2}{g}+1)$	$N^2(\frac{k^2}{b^2}+\frac{2}{b})$	$N^2(\frac{k^2}{gb^2}+\frac{2}{b})$
BN Params	$4N$	$8N$	$N(2+\frac{4}{b})$	$N(2+\frac{4}{b})$

each collection of the form A_i into a vector, attention transfer involves learning the student network by minimising

$$\mathcal{L}_{AT} = \mathcal{L}_{CE} + \beta \sum_{i=1}^{L} \left\| \frac{\mathbf{f}(A_i^t)}{||\mathbf{f}(A_i^t)||_2} - \frac{\mathbf{f}(A_i^s)}{||\mathbf{f}(A_i^s)||_2} \right\|_2, \quad (1)$$

where β is a hyperparameter, and $\mathcal{L}_{CE}$ is the standard cross-entropy loss. In [56] the authors use $\mathbf{f}(A_i) = (1/N_{A_i}) \sum_{j=1}^{N_{A_i}} \mathbf{a}_{ij}^2$, where N_{A_i} is the number of channels at layer i.

III. SUBSTITUTING FULL CONVOLUTIONS

Here, we consider a range of alternatives for replacing full $k \times k$ convolutions (where k is invariably 3) to form compressed student networks. These are prevalent in many popular and widely-used network architectures [4], [5], [31], [58]. Specifically, we focus on replacing the convolutional block structure present in residual networks.

First, consider a standard two dimensional convolution that contains N_{out} filters, each of size $N_{\text{in}} \times k \times k$. N_{out} is the number of channels of the layer output, N_{in} is the number of channels of the input, and $k \times k$ is the kernel size of each convolution. In modern networks it is almost always the case that $N_{\text{in}} \leqslant N_{\text{out}}$. Let $N = \max(N_{\text{in}}, N_{\text{out}})$. Then the parameter cost of this layer is $N_{\text{in}} N_{\text{out}} k^2$, and is bounded by N^2k^2. In a typical residual network, a block contains two such convolutions. We will refer to this as a *Standard* block S, and it is outlined in Table 1.

An alternative approach is to separate each convolution into g groups. By restricting the convolutions to only mix channels within each group, we obtain a substantial reduction in the number of parameters for a grouped computation: for example, for $N_{\text{in}} = N_{\text{out}} = N$ the cost changes from N^2k^2 for a standard layer to g groups of $(\frac{N}{g})^2 k^2$ parameter convolutions, hence reducing the parameter cost by a factor of g. This is illustrated in Figure 1. We can then provide some cross-group mixing by following each grouped convolution with a pointwise convolution, with a N^2 parameter cost (when $N_{\text{in}} \neq N_{\text{out}}$ the change in channel size occurs across this pointwise convolution). We refer to this substitution operator as $G(g)$ (grouped convolution with g groups).

In the original ResNet paper [5] the authors introduced a bottleneck block which we have parameterised, and denoted as $B(b)$ in Table 1: the input first has its channels decreased by a factor of b via a pointwise convolution, before a full convolution is carried out. Finally, another pointwise convolution brings the representation back up to the desired N_{out}. We can reduce the parameter cost of this block further by replacing the full convolution with a grouped one; the *Bottleneck Grouped + Pointwise* block is referred to as $BG(b, g)$.

These substitute blocks are compared in Table 1 and their computational costs are given. In practice, by varying the bottleneck size and the number of groups, network parameter numbers may vary over two orders of magnitude; enumerated examples are given in Table 2.

Using grouped convolutions and bottlenecks are common methods for parameter reduction when designing a network architecture. Both are easy to implement in any deep learning framework. We demonstrate empirically in Section V-A that using these substitutions with effective model distillation allows for substantial compression with minimal reduction in performance.

IV. SUBSTITUTING POINTWISE CONVOLUTIONS

It is increasingly becoming the case that most of the parameters in a network are used for pointwise (1×1) convolutions. For example, it is common to use pointwise convolutions to modulate the channel dimension [7], [59]. Separable convolutions—which consist of a grouped convolution followed by a pointwise convolution—are one of the elementary operations in the neural architecture search space of DARTS [23]. Indeed, the substitutions we used for full convolutions in Section III introduce pointwise convolutions.

A pointwise convolution is equivalent to a fully-connected linear layer applied at each pixel location of an image, or its downstream representation. Because of this, we can take advantage of techniques developed to compress linear layers; we can form student networks by substituting out the pointwise convolutions present in our networks for cheaper alternatives to achieve high performance for reduced parameter counts and computational cost.

TABLE 2. Student network test error on CIFAR-10/100. Each network is a WideResNet with its depth-width (D-W) given in the first column, and with its block type (corresponding to Table 1 in the second. *N* refers to the channel width of each block, and *M* refers to the channel width after the bottleneck where applicable. The total parameter cost of the networks for CIFAR-10 is given, as well as the number of mult-add operations they use. Note that CIFAR-100 networks use an extra 11.6K parameters and mult-adds over their CIFAR-10 equivalents as they have a larger linear classification layer. Errors are reported for (i) learning with no distillation i.e. from scratch (Scr), (ii) knowledge distillation with a teacher (KD), and attention transfer with a teacher (AT). The same teacher is used for training, and is given in the first row. This table shows that (i) through attention transfer it is possible to cut the number of parameters of a network, but retain high performance and (ii) for a similar number of parameters, students with cheap convolutional blocks outperform those with expensive convolutions and smaller architectures.

				CIFAR-10			CIFAR-100		
D-W	Block	Params (K)	MAdds (M)	Scr	KD	AT	Scr	KD	AT
T 40-2	S	2243.5	328.3	4.79	–	–	23.85	–	–
16-2	S	691.7	101.4	6.53	6.03	5.66	27.63	27.97	27.24
40-1	S	563.9	83.6	6.48	6.39	5.50	29.64	30.21	28.24
16-1	S	175.1	26.8	8.81	8.75	7.72	34.00	37.28	33.74
40-2	S-2x2	1007.1	147.4	5.89	6.03	5.09	27.20	26.98	26.09
40-2	G(2)	1359.0	198.1	5.30	5.37	4.87	25.94	24.92	24.45
40-2	G(4)	814.7	118.5	5.50	5.81	5.00	26.20	25.48	25.30
40-2	G(8)	542.5	78.7	5.92	5.72	5.05	26.49	26.64	25.71
40-2	G(16)	406.4	58.8	6.65	6.38	5.13	28.85	27.10	26.34
40-2	G(N/16)	641.3	133.9	5.72	5.72	5.12	27.08	26.11	25.78
40-2	G(N/8)	455.8	86.4	6.07	5.61	5.06	27.85	27.05	26.15
40-2	G(N/4)	363.1	62.6	6.93	6.45	5.31	28.91	27.93	26.85
40-2	G(N/2)	316.7	50.8	7.12	6.83	5.98	30.24	28.89	28.54
40-2	G(N)	293.5	44.8	8.51	8.01	6.57	31.84	29.99	30.06
40-2	B(2)	431.8	64.5	6.36	6.28	5.37	28.27	28.08	26.68
40-2	B(4)	150.9	22.8	7.94	7.83	6.93	31.63	33.63	30.56
40-2	BG(2,2)	286.7	43.3	6.12	6.25	5.57	28.51	28.82	28.28
40-2	BG(2,4)	214.1	32.7	6.75	6.75	6.05	29.39	29.25	28.54
40-2	BG(2,8)	177.8	27.3	6.94	6.98	6.09	30.21	29.34	28.89
40-2	BG(2,16)	159.7	24.7	6.77	6.97	6.19	30.57	30.54	29.46
40-2	BG(2,M/16)	238.3	46.8	6.26	6.50	6.02	29.69	28.69	29.05
40-2	BG(2,M/8)	189.9	34.4	6.75	6.49	5.94	29.09	29.13	28.16
40-2	BG(2,M/4)	165.7	28.2	7.06	7.15	6.03	30.42	30.28	28.60
40-2	BG(2,M/2)	153.6	25.1	7.45	7.47	6.17	30.44	30.66	29.51
40-2	BG(2,M)	147.6	23.6	7.95	7.99	6.67	30.90	31.18	30.03
40-2	BG(4,M)	81.4	13.0	9.04	8.61	7.87	33.64	37.34	32.89

The various substitutions considered are described in Section IV-A. Each uses either fewer parameters, fewer mult-adds or both. These are by no means exhaustive, and are representative examples of a general approach. To make training these layers practical, we account for the effect of using compressed linear layers in the choice of weight decay. We derive the weight decay parameters required to stabilise training in Section IV-B.

A. SUBSTITUTE EFFICIENT LINEAR TRANSFORMS

Here, we describe methods that we use to replace pointwise convolutions to construct compressed student networks. All provide an approximation to the operation of a dense random matrix in a linear layer: a matrix-vector product of that matrix with an input vector of the form $\mathbf{y} = \mathbf{W}\mathbf{x}$, where $\mathbf{y}$ is the output vector, $\mathbf{W}$ is the dense random matrix, and $\mathbf{x}$ is the input vector.

1) ACDC

In [30], $\mathbf{W}$ is decomposed into a stack of L ACDC layers:

$$\mathbf{W} = \prod_{l=1}^{L} \mathbf{A}_l \mathbf{C} \mathbf{D}_l \mathbf{C}^{-1} \mathbf{P} \tag{2}$$

where $\mathbf{A}$ and $\mathbf{D}$ are diagonal matrices, $\mathbf{C}$ and $\mathbf{C}^{-1}$ are forward and inverse discrete cosine transforms, and $\mathbf{P}$ is a random permutation matrix. As the operation of a random permutation may be time consuming, we replace $\mathbf{P}$ with a *riffle shuffle*. A riffle shuffle is a fixed permutation, splitting the input in half and then interleaving the two halves. This was found to work as well as a fixed random permutation and can be evaluated much faster as observed by [38]. For $\mathbf{W} \in \mathbb{R}^{N \times N}$ the computational complexity is $O(N \log N)$ and storage cost is $O(N)$ [30].

2) TENSOR-TRAIN

First, we assume it is possible to map a higher dimensional tensor to our weight matrix using a reshape operation: $\mathbf{y} = \mathbf{W}\mathbf{x} = \text{reshape}^{\mathbb{R}^{N \times N}}(\mathcal{A})\mathbf{x}$. This allows us to use a tensor decomposition to represent $\mathcal{A}$ and implement the linear transform using fewer parameters. In Tensor-Train (TT) [60], $\mathcal{A}$ is decomposed as:

$$\mathcal{A}(i_i, \ldots, i_d) = \mathbf{G}_1(i_1) \ldots \mathbf{G}(i_d) \tag{3}$$

where $\mathbf{G}_k(i_k)$ are $r_{k-1} \times r_k$ matrices, with the boundary conditions that ensure $r_0 = r_d = 1$. Each element of the tensor can then be reproduced by performing this sequence of matrix products.

The parameter savings using this method depend on the number of dimensions possessed by the tensor storing the weights. It is possible to perform a matrix-vector, or matrix-matrix, product between two TT tensors. Alternatively, as in our experiments, we can compute the weight matrix from the $\mathbf{G}_k$ factors and backpropagate the error to update those factors with automatic differentiation.

In our experiments we found it best to reshape weight matrices to 3 dimensions, with approximately equal sizes. We then set the TT-rank $r_1, \ldots, r_{d-1}$ to control the level of compression. This is in line with previous work substituting TT tensors into fully connected layers of deep neural networks [29].

3) TUCKER DECOMPOSITION

The Tucker decomposition also decomposes a tensor $\mathcal{A} \in \mathbb{R}^{I_0,\ldots,I_d}$, but in this case uses a low rank core $\mathcal{G} \in \mathbb{R}^{R_0,\ldots,R_d}$ projected by factors $\mathbf{U}_k \in \mathbb{R}^{R_k,I_k}$ [61]:

$$\mathcal{A} = \mathcal{G} \times_0 \mathbf{U}_0 \ldots \times_d \mathbf{U}_d \tag{4}$$

where $\times_k$ denotes the *k-mode product*: a matrix product on dimension k while casting over the remaining dimensions. In our experiments, to compare with TT, we only use the Tucker decomposition to store our weight matrices. As with the TT decomposition, we compute the weight matrix, then backpropagate gradients in order to update the $\mathbf{U}_k$ factors.

4) RANK-FACTORISED (RF) DECOMPOSITION

A rank factorised matrix is a linear bottleneck. It is chosen as a baseline against which to compare methods from the literature. In place of a dense matrix we first map an input to a smaller number of dimensions, and then back to the output number of dimensions. This uses two weight matrices $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{bn}} \times d_{\text{in}}}$ and $\mathbf{W}_2 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{bn}}}$, where the input dimensionality is d_{in}, bottleneck is d_{bn} and output is d_{out}. The linear transformation from an input $\mathbf{X}$ to an output $\mathbf{Y}$ can then be expressed as $\mathbf{y} = \mathbf{W}\mathbf{x} = \mathbf{W}_2\mathbf{W}_1\mathbf{x}$.

This parameterisation can be implemented in popular deep learning frameworks with two linear layers in sequence, and can give significant efficiency benefits. For $d_{\text{in}} = d_{\text{out}} = d$, the number of parameters used by applying a dense weight matrix $\mathbf{W}$ to an input vector is d^2, while the total parameters used in $\mathbf{W}_1$ and $\mathbf{W}_2$ is $\frac{2\,d^2}{b}$ where $b = \frac{d}{d_{\text{bn}}}$.

5) HASHEDNET

A virtual weight matrix $\mathbf{V}$ is built from real weights $\mathbf{w}$ using a hash function $\mathbf{h}$ to index those weights:

$$y_i = \sum_{j=1}^{N} V_{ij} x_j = \sum_{j=1}^{N} w_{h(i,j)} x_j \tag{5}$$

HashedNets use a hash function to retrieve the weights used in their network [28]. The particular hash function used in this case takes as input indices in the *virtual* weight matrix, $\mathbf{V}$, used in the linear transformation, and produces as output a single index into a set of *real* weights $\mathbf{w}$. As the compression only depends on the number of virtual weights we choose to use, this method is extremely flexible for storage compression. Note that some virtual weights may not be used, as described in Appendix A.

6) LINEARISED SHUFFLENET

The Shufflenet unit [38] can be related to circulant transforms like ACDC [30] by defining a generalised UGConv block [62]. Unlike applications of circulant transforms, these units are used to implement a state-of-the-art image classification architecture (ShuffleNet). We propose a linear version of the unit, represented by:

$$\mathbf{y} = \mathbf{W}\mathbf{x} = \mathbf{B}_2\mathbf{P}\mathbf{B}_1\mathbf{x} \tag{6}$$

where $\mathbf{B}_1$ and $\mathbf{B}_2$ are block diagonal matrices implemented by grouped 1×1 convolutions, and $\mathbf{P}$ is a permutation implemented by a riffle shuffle. While this was not proposed in the literature as a method to compress a linear transformation, the building blocks involved are similar to those used in the ACDC structured efficient linear transformation and so we consider it here.

B. COMPRESSION RATIO SCALED WEIGHT DECAY

One way to motivate L2 regularisation in neural networks is to say that it is equivalent to MAP inference with a Gaussian prior on the weights [63, p.225]. In the context of changing layer structure, we would wish to preserve the total variance of the weight matrix prior under the layer replacement. As the number of parameters tends towards the number in the full weight matrix, we will then tend toward the original weight decay factor. Under the simplifying assumption that the weights, $\{w_n\}_{n=1}^{N}$ are Gaussian distributed with variance $\frac{1}{\sqrt{d}}$, where d is the weight decay factor, then total variance is:

$$\sum_{n=1}^{N} E\left[w_n^2\right] = \sum_{n=1}^{N} \frac{1}{d} E\left[z^2\right], \tag{7}$$

where z is Gaussian distributed with variance 1. Then we have:

$$\sum_{n=1}^{N} Var(w_n) = \frac{N}{d}. \tag{8}$$

Hence, to maintain total variance for M parameters in the compressed layer, we should use the scaled weight decay term:

$$d_c = \frac{Md}{N}. \tag{9}$$

In practice this means multiplying the weight decay factor for compressed weight matrices by the compression ratio M/N. We demonstrate in Appendix C: in Figure 10, we illustrate that this stabilises training and improves performance. It has the desirable property of providing a smooth interpolation to an uncompressed matrix – where the weight decay would be the default. We refer to this approach as *compression ratio scaled* (CRS) weight decay.

V. EXPERIMENTS

In Section III we proposed suitable cheap alternatives for full convolutions that may be used with our approach. Similarly, in Section IV we proposed efficient linear layers that may be used as replacements for pointwise convolutions.

Here, we experimentally verify our approach and show how such substitutions, combined with distillation, allow us to perform substantial network compression at a minimal loss of accuracy. In Section V-A we present experiments for substituting full convolutions, and in Section V-B we present experiments for pointwise substitution.

A. SUBSTITUTING FULL CONVOLUTIONS

In Section V-A1 we conduct experiments on the CIFAR-10 and CIFAR-100 datasets (10/100-way classification of 32×32 images). We take a WideResNet [58] teacher and construct students with (i) the cheap substitute blocks of Section III, (ii) student networks with smaller architectures (i.e. fewer layers/filters) as a baseline. We distil with (i) knowledge distillation [25] and (ii) attention transfer [56]. We also train the networks without any form of distillation (i.e. from scratch) to observe whether the distillation process is necessary to obtain good performance. In this way we demonstrate that the high performance comes from the distillation, and cannot be achieved by directly training the student networks using the data. Then, in Section V-A2 we apply our approach to ResNets on the challenging ImageNet dataset (1000-way classification of 224×224 images).

1) CIFAR EXPERIMENTS

For our experiments we utilise the Wide Residual Network (WRN) architecture [58]; the bulk of the network lies in its $\{conv2, conv3, conv4\}$ groups and the network depth d determines the number of convolutional blocks n in these groups as $n = (d-4)/6$. The network width, denoted by k, affects the channel size of the filters in these blocks. Note that when we employ attention transfer the student and teacher outputs of groups $\{conv2, conv3, conv4\}$ are used as $\{A_1, A_2, A_3\}$ in the second term of Equation (1) with $N_L = 3$.

For our teacher network we use WRN-40-2 (a WRN with depth 40 and width multiplier 2) with standard (S) blocks. 3×3 kernels are used for all non-pointwise convolutions in our student and teacher networks unless stated otherwise. For our student networks we use:

- WRN-40-1, 16-2, and 16-1 with S blocks. These are student networks that are thinner and/or more shallow than the teacher and represent typical student networks.
- WRN-40-2 with S blocks where the 3×3 kernels have been replaced with 2×2 dilated kernels (as described in [64]). This allows us to see if it possible to naively reduce parameters by effectively zeroing out elements of standard kernel.
- WRN-40-2 using a bottleneck block B with $2\times$ and $4\times$ channel contraction (b).
- WRN-40-2 using a grouped + pointwise block G for group sizes (g) {2, 4, 8, 16, N/16, N/8, N/4, N/2, N} where N is the number of channels in a given block. This allows us to explore the spectrum between full convolutions ($g = 1$) and fully separable convolutions ($g = N$).
- WRN-40-2 with a bottleneck grouped + pointwise block BG. We use $b = 2$ with groups sizes of {2, 4, 8, 16, M/16, M/8, M/4, M/2, M} where $M = N/b$ is the number of channels *after the bottleneck*. We use this notation so that $g = M$ represents fully separable convolutions and we can easily denote divisions thereof. $BG(4, M)$ is also used to observe the effect of extreme compression.

For training we used minibatches of size 128. Before each minibatch, the images were padded by 4×4 zeros, and then a random 32×32 crop was taken. Each image was left-right flipped with a probability of one half. Networks were trained for 200 epochs using SGD with momentum fixed at 0.9 with an initial learning rate of 0.1. The learning rate was reduced by a factor of 0.2 at the start of epochs 60, 120, and 160. For knowledge distillation we set α to 0.9 and used a temperature of 4. For attention transfer β was set to 1000.

Figure 2 compares the parameter cost of each student network (on a log scale) against the test error on CIFAR-10 obtained with attention transfer. On this plot, the ideal network would lie in the bottom-left corner (few parameters, low error). Almost every network with the same architecture as the teacher, but with cheap convolutional blocks, performs better for a given parameter budget than the reduced architecture networks with standard blocks. $BG(2, 2)$ outperforms 16-2 (5.57% vs. 5.66%) despite having considerably fewer parameters (287K vs. 692K). Several of the networks with BG blocks both significantly outperform 16-1 and use fewer parameters.

It is encouraging that significant compression is possible with only small losses; several networks perform almost as well as the teacher with considerably fewer parameters – $G(N/8)$ has an error of 5.06%, close to that of the teacher, but has just over a fifth of the parameters. $BG(2, M/8)$ has less than a tenth of the parameters of the teacher, for a cost of 1.15% increase in error. Even simply switching all convolutions with smaller, dilated equivalents ($S - 2 \times 2$) allows one to use half the parameters for a similar performance.

Note that grouped + pointwise convolutions are often used in their fully separable [34] form. However, the networks with half, or quarter that number of groups perform substantially better for a modest increase in parameters. $G(N/4)$ has 363K parameters compared to the 294K of $G(N)$ but has an error that is 1.26% lower. The number of groups is an easy parameter to tune to trade some performance for a smaller network. Grouped + pointwise convolutions also work well in conjunction with a bottleneck of size 2, although for large bottlenecks the error increases significantly, as can be seen for $BG(4, M)$. Despite this, it is still of comparable performance to 16-1 with half the parameters. Similar trends are observed for CIFAR-100 in Table 2b.

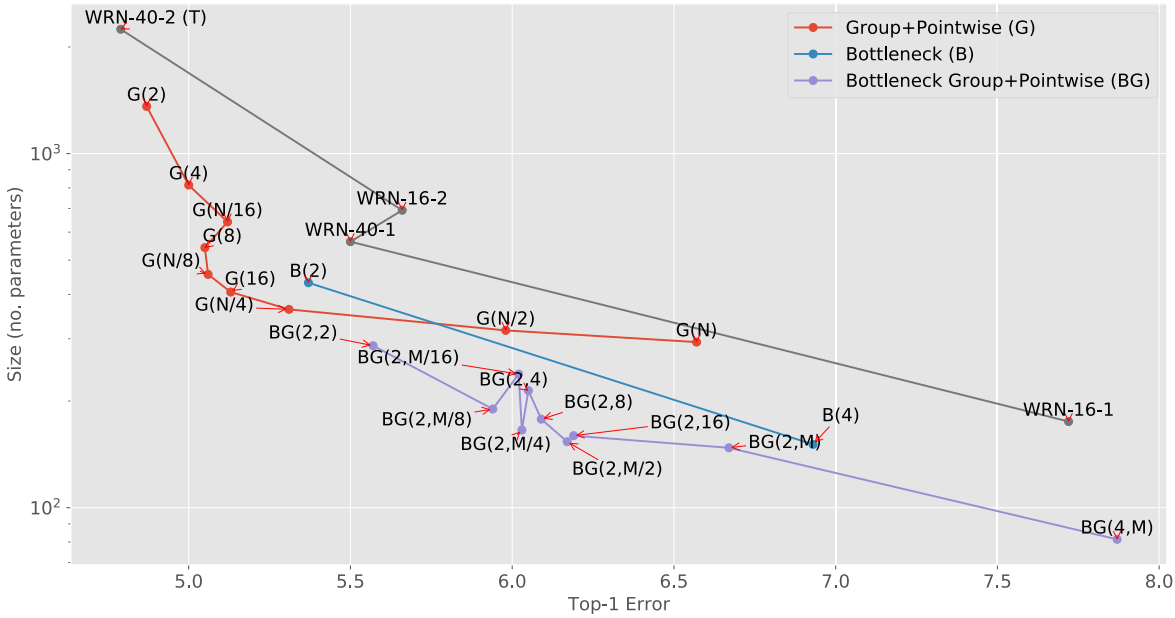

FIGURE 2. Test error vs. No. parameters for student networks learnt with attention transfer on CIFAR-10. Note that the x-axes are log-scaled. Points on the red curve correspond to networks with *S* convolutional blocks and reduced architectures. All other networks have the same WRN-40-2 architecture as the teacher but with cheap convolutional blocks: *G*, *B*, and *BG*. The blocks are described in Table 1. Notice that the student networks with cheap blocks outperform those with smaller architectures and standard convolutions for a given parameter budget.

We also observe that training a student with attention transfer is substantially better than using knowledge distillation, or simply training from scratch. Consider Table 2, which shows the attention transfer errors of Figure 2 (the AT column) alongside those of networks trained with knowledge distillation (KD), and no distillation i.e. from scratch (Scr) for CIFAR-10 and CIFAR-100. In all cases, the student network trained with attention transfer is better than the student network trained by itself – the distillation process appears to be necessary. Some performances are particularly impressive; on CIFAR-10, for $G(2)$ blocks the error is only 0.08% higher than the teacher despite the network having 60% of the parameters.

These results support our claim that greater model compression through distillation is possible by substituting the convolutional blocks in a network, rather than by shrinking its architecture. We have also demonstrated that the blocks outlined in Table 1 are suitable substitutes. By observing Table 2 we can also see that our networks with cheap substitute blocks utilise fewer mult-add operations than their standard equivalents, which roughly corresponds to a faster runtime. Note that runtime on a given platform or device is dependent on specifics (memory paging, choice of libraries etc.), so mult-adds are not always fully indicative of runtime, but are a decent approximation in a platform/implementation-agnostic setting.

2) IMAGENET

We use a pre-trained ResNet-34 [5] (21.8M parameters) as a teacher and we train several networks using attention transfer (AT). We compare student networks that have the architecture of ResNet-34, with *cheaper* convolutions, to those that have reduced architectures, and full convolutions. Note, that the bulk of the parameters in a ResNet are contained in four groups, as opposed to the three of a WideResNet. We train the following student networks: (i) ResNet-18, (ii) ResNet-18 with the channel widths of the last three groups halved (Res18-0.5), ResNet-34 with each convolutional block replaced by (iii) a $G(N)$ block and (iv) a $G(4)$ block. Validation errors for these networks are available in Table 3.

Consider Res34-G(N) and Res18-0.5, which both have roughly the same parameter cost (∼3M). After distillation, the former has a significantly lower top-5 error (10.66% vs. 15.02%). This again supports our claim that is preferable to cheapen convolutions, rather than shrink the network architecture. Res34-G(N) trained from scratch has a noticeably higher top-5 error (12.26%), it benefits from distillation. Conversely, distillation makes Res18-0.5 slightly worse, suggesting that it has no further representational capacity.

Res34-G(4) similarly outperforms Res18 (these are roughly similar in cost at 8.1M and 11.7M parameters respectively), although in this case the latter does benefit from distillation. It is intriguing that Res34-G(4) trained from scratch is actually on par with the original teacher (having a 0.12% lower top-1 error, and a 0.05% higher top-5 error) despite having 13 million fewer parameters; this generalisation capability of grouped convolutions in networks has been observed previously by [41]. Distillation is able to push its performance slightly further to the point that its top-5 error surpasses that of the teacher (8.43% vs. 8.57%).

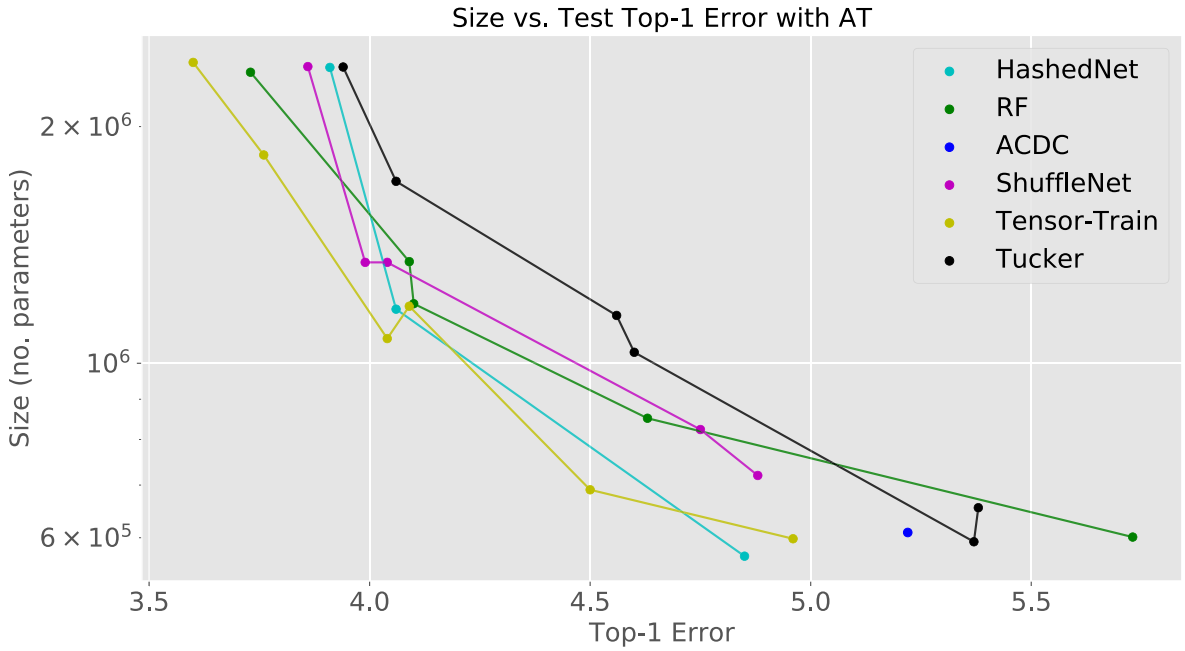

FIGURE 3. The relationship between top-1 error on the validation set and the number of parameters is plotted for experiments involving WRN-28-10 on CIFAR-10, for experiments using AT. Each substitute linear transform tested is indicated in the legend. On this problem, both Tensor-Train and HashedNet substitutions are able to achieve the highest rates of compression while maintaining performance. At lower compression settings, all methods compared achieve comparable top-1 error.

TABLE 3. Top 1 and top 5 classification errors (%) on the validation set of ImageNet for models (i) trained from scratch, and (ii) those trained with attention transfer with ResNet-34 (Res34) as a teacher. Res18 refers to a Resnet-18, and Res18 w/2 is a Resnet-18 where the channel width in the last three groups is halved. Res34 G(*x*) is a ResNet-34 with each convolutional block replaced by a G(*x*) block. We are able to produce compressed, high-performing networks on this challenging dataset.

		Scratch		AT	
Model	Params	Top 1	Top 5	Top 1	Top 5
Res34 **T**	21.8M	26.73	8.57	–	–
Res18	11.7M	30.36	11.02	29.18	10.05
Res34 G(4)	8.2M	26.61	8.62	26.58	8.43
Res18 w/2	3.2M	36.96	15.01	37.20	15.02
Res34 G(N)	3.2M	32.98	12.26	30.16	10.66

3) SUMMARY

- We have applied our method to WideResNets for a multitude of substitute blocks on CIFAR, all of which allow for significant compression with minimal loss of accuracy e.g. with $G(N/8)$ we get $5\times$ compression for only 0.27% change in accuracy on CIFAR-10.
- We have demonstrated that our findings hold for ImageNet, where we were able to compress a ResNet by $2.65\times$ and see its accuracy increase.

B. SUBSTITUTING POINTWISE CONVOLUTIONS

We train student networks where each pointwise (1×1) convolution in the original teacher network is substituted for a particular linear transform from Section IV-A. These transforms are: ACDC, Tensor-Train, Tucker decomposition, Rank-factorised (RF) decomposition, HashedNet, and Linearised ShuffleNet. Each of these substitutions has a tuning parameter that can be altered to determine the number of parameters utilised, allowing us to compare networks for a range of parameter budgets. We perform experiments on CIFAR-10 [65] with (i) Wide-ResNets (WRN) [58], specifically WRN-28-10, and (ii) the network discovered through differentiable architecture search (DARTS) [23] as teacher networks. We also experiment on ImageNet [66] with (i) WRN-50-2 and (ii) MobileNetV2 as teacher networks. We chose parameter budgets over which networks with each substitution under consideration would have support (see Appendix B). When training:

1) We perform *attention transfer* (AT) [56] on each substitute network with a trained version of the original network as a teacher.
2) We utilise *CRS weight decay*, as defined in Section IV-B.

1) WRN-28-10 ON CIFAR-10

The base WRN-28-10 teacher network achieves a top-1 validation error of 3.2% and has 36.5M parameters. We produce substitute student networks at three approximate parameter budgets: 2.4M, 1.2M, and 0.6M. Note that these correspond to very high compression rates.

Each network was trained for 200 epochs with a learning rate starting at 0.1 and scaled by 0.2 on epochs 60, 120 and 160. Momentum was set to 0.9 and the minibatch size was 128. Weight decay was set to 5×10^{-4} and scaled in all experiments according to the method described in

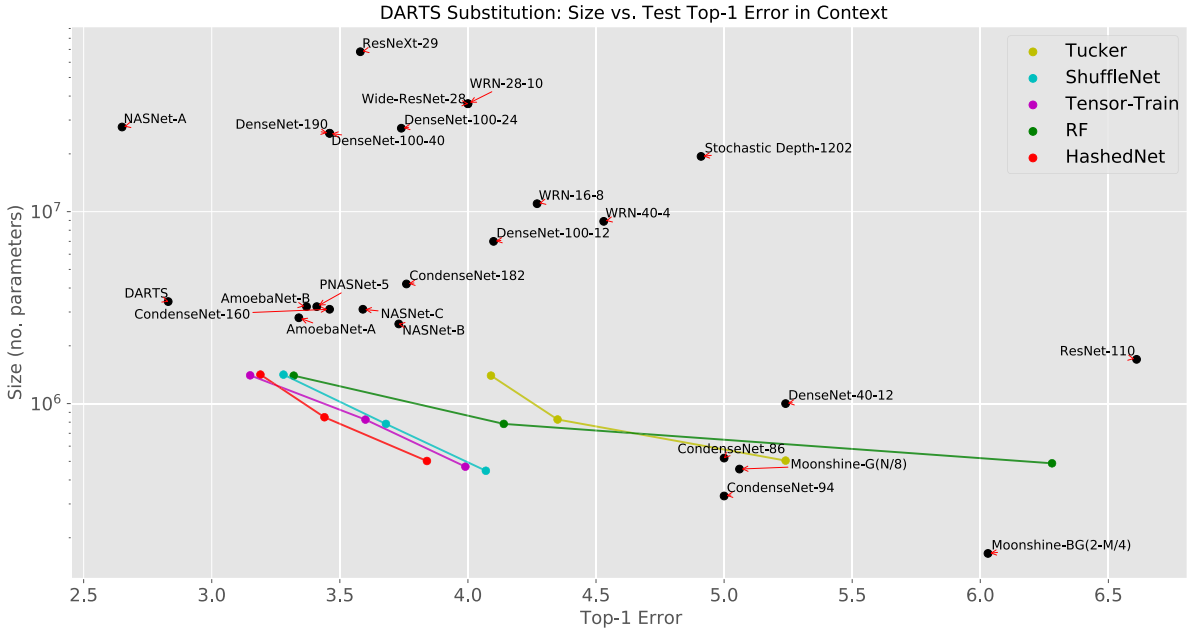

FIGURE 4. Top-1 validation errors on CIFAR-10 for DARTS networks with substitute linear transforms and their parameter totals. Each substitute linear transform tested is indicated in the legend. ACDC is omitted as it failed to converge below 8% in any case. We compare against recent networks presented in the literature, including: DenseNet [7], Moonshine [27], Wide ResNet [58], ResNeXt [39], Stochastic Depth [67], GoogleNet [68], CondenseNet [42], NASNet [69], ResNet [5], PNASNet [70], AmoebaNet [71] and DARTS [23]. Using these substitutions we are able to explore a new region in the Pareto Frontier.

Section IV-B. Data was augmented with random crops, left-right flips and Cutout [72].

The relationship between the number of parameters used by our substitute WRN-28-10 networks and the top-1 error—through AT with the teacher network—is illustrated in Figure 3. We report mult-adds where possible in Figure 5.[1]

One might expect HashedNet or Tensor-Train to work best as compression methods, as they do not necessarily reduce the number of mult-adds used by the network. HashedNet, in particular, substitutes a weight matrix of precisely the same size at test time, and applying that weight matrix uses the same number of mult-adds used by the original network. While all of the considered methods produce a network that contains less than 10% the parameters used by the teacher network while losing only 1% error, HashedNet and Tensor-Train substitutions can maintain this error using **less than 3% of the parameters**. Unfortunately, ACDC was only able to place a single point at the lowest compression ratio. It performs comparably well, but becomes unstable with larger numbers of ACDC layers.

2) DARTS NET ON CIFAR-10

DARTS net is a highly competitive image classification network [23] obtained through neural architecture search [22], [69] achieving 2.83% error while using only 3.8M parameters. We use this as a teacher network, and substitute its pointwise convolutions (present in separable convolutions) for parameter budgets of 1.42M, 0.83M, and 0.49M.

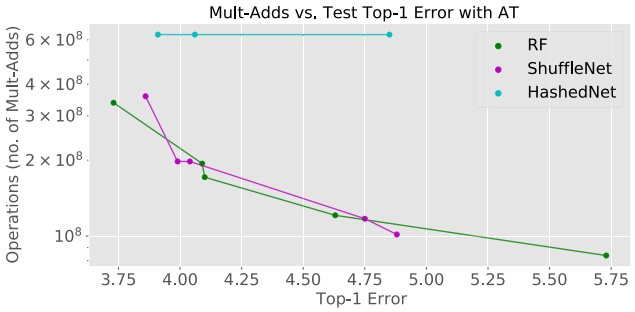

FIGURE 5. The relationship between top-1 error on the validation set and the number of mult-adds consumed by each network is plotted for experiments involving WRN-28-10 on CIFAR-10.

Each network was trained with AT for 600 epochs using a cosine annealed learning rate schedule starting at 0.025. Momentum was set to 0.9 and the minibatch size was 96. Weight decay was set to 3×10^{-4} and scaled in all experiments according to the method described in Section IV-B. The auxiliary classification head was used in training, but not counted at test time, and the drop-path method from the paper [23] followed the same schedule of a linear increase in drop probability from 0 to 0.2 over the learning schedule. Data was again augmented with random crops, left-right flips and Cutout [72].

The validation errors of these substitute networks against their parameter total is shown in Figure 4. We can achieve

[1]We were unable to calculate robust estimates for the mult-adds used by TT or Tucker substitutions, as they would depend on the choice of rounding and efficient matrix-vector multiplication algorithms used [60].The only stable ACDC experiment cost 1.8×10^9 mult-adds, far more than competing methods.

TABLE 4. Top-1 validation errors on ImageNet for WRN-50-2 with our proposed substitutions: ShuffleNet, Tensor-Train and RF. Each method is tested at two approximate parameter budgets. Compression is given as a percentage of the original model size. Methods from the literature are provided for comparison: WRN-50-2 [58], ShuffleNet [38], DenseNet [7], MobileNet [35], ACDC [30] and TT [29]. These results show that our method generalises to large, deep convolutional neural networks for image classification.

Model	Substitution	Parameters	Mult-Adds	Top-1 (%)	Compression (%)	
					Parameters	Mult-Adds
WRN-50-2	ShuffleNet	6.04M	0.91G	29.7	8.77	8.00
WRN-50-2	ShuffleNet	17.72M	3.22G	26.9	25.72	28.22
WRN-50-2	RF	4.35M	0.53G	39.8	6.3	4.62
WRN-50-2	RF	17.55M	2.83G	25.4	25.5	24.77
WRN-50-2	HashedNet	4.35M	4.86G	33.5	6.32	42.59
WRN-50-2	HashedNet	17.61M	4.86G	24.5	25.56	42.59
WRN-50-2	Tensor-Train	4.34M	4.86G	33.2	6.30	42.59
WRN-50-2	Tensor-Train	17.58M	4.86G	24.9	25.52	42.59
WRN-50-2	None	68.9M	11G	21.9	-	-
ShuffleNet	None	1.87M	0.14G	32.4	-	-
ShuffleNet 2×	None	7.51M	0.53G	24.7	-	-
DenseNet-121	None	9M	6G	25.0	-	-
MobileNet	None	4.2M	0.57G	29.4	-	-
CaffeNet	ACDC	9.7M	n/a	43.3	16.7	n/a
VGG-16	Tensor-Train	18.65M	n/a	32.2	13.5	n/a
VGG-19	Tensor-Train	24.0M	n/a	31.6	16.7	n/a

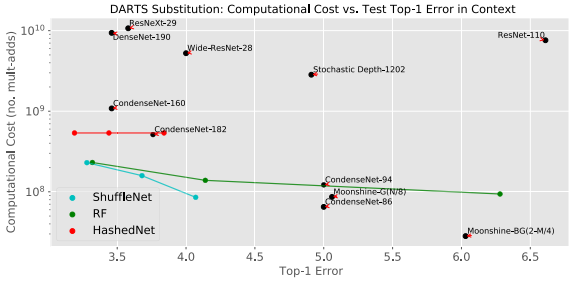

FIGURE 6. Top-1 validation errors on CIFAR-10 for DARTS networks with substitute linear transforms and their associated mult-add cost. Not all substitute transforms tested could be included here, as noted in Figure 5, not all could be easily estimated. We compare against recent networks in the literature, including: ResNext [39], DenseNet [7], Wide ResNet [58], CondenseNet [42], Stochastic Depth [67], and ResNet [5]. On this Figure, ***ShuffleNet substitutions appear to be Pareto optimal***, but we were not able to compare against a large number of papers that do not report mult-add cost on CIFAR-10.

compression of up to 20% of the original number of parameters for HashedNet, ShuffleNet and Tensor-Train substitutions while still being *within 1% the original top-1 error*.

We can see that our networks explore an *empty region of the Pareto frontier* in the context set by the literature. The top-1 error achieved through a HashedNet substitution is equal to or lower than all published networks compared against, save for DARTS and NASNet-A, while using several times fewer parameters.

When we compute the mult-adds used by these networks (Figure 6) we observe similar trends. Notably, our ShuffleNet substitution performs extremely similarly to the original network while using around 5 times fewer operations. In terms of mult-adds, these networks again extend the Pareto boundary defined by all methods considered.

3) WRN-50-2 ON IMAGENET

Based on their performance in the two CIFAR-10 experiments, we chose HashedNet, Tensor-Train and ShuffleNet to compare on ImageNet with WRN-50-2 as a teacher network. We also included RF substitution as a baseline.

Each network was trained for 90 epochs with a learning rate of 0.1 scaled by 0.1 at epochs 30 and 60. Momentum was set to 0.9 and the minibatch size was 256. Weight decay was 1×10^{-4} and scaled according to the method described in Section IV-B. Data was augmented with random crops and left-right flips.

The results of the experiments are shown in Table 4. ImageNet is a more difficult problem than CIFAR-10, and we see that performance rapidly degrades as we reduce the number of parameters, although this appears to be the same trend observed with published networks in the literature. The compression rates achieved with our agnostic substitutions compare favourably to other state-of-the-art image classification networks in the field. This demonstrates the generalisation of this method to even the largest deep convolutional neural networks for image classification.

4) MOBILENETV2 ON IMAGENET

The results on ImageNet in Table 4 show significant compression of WRN-50-2, but do not show that we can maintain performance at the state of the art. To investigate this, we apply a ShuffleNet substitution to a MobileNetV2 [59] network and train it according to the published regime. The top-1 error of the resulting network is compared against competitive compressed networks in terms of both parameter count and mult-add count in Figure 7. The trained network is competitive in the space of networks of fewer than 1 million parameters, for example outperforming ShuffleNet.

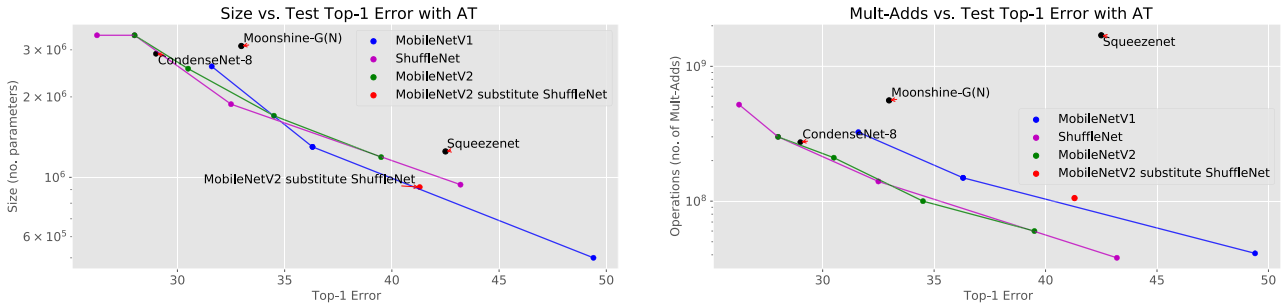

FIGURE 7. Substitution of a linear ShuffleNet block in a MobileNetV2 [59].

5) SUMMARY

- We have applied our method to WRN-28-10 and the DARTS network for a host of linear substitutions on CIFAR. This allowed us to significantly compress these networks while maintaining a competitive performance. For example, we were able to compress WRN-28-10 by 30× for less than a percent change in accuracy.
- We veirfied that these results held on ImageNet to show the generalisation capability of our approach.

VI. CONCLUSION

After training a large, deep model it may be prohibitively time consuming to adopt and tune a model compression strategy in order to deploy it. We have demonstrated a model compression strategy that is fast to apply, and does not require additional engineering. We have demonstrated that full convolutions may be substituted for e.g. grouped convolutions to produce compressed student networks. For substituting pointwise convolutions we can leverage efficient dense layer and trivially stabilise the learning dynamics by using CRS weight decay; allowing us to explore a new region of the Pareto frontier for both parameter and computational cost on CIFAR-10. In this paper we have considered uniform substitutions for simplicity, but it is possible to mix different substitutions within a network for even greater performance [73].

APPENDIX A

HASHEDNET DISCONNECTED WEIGHTS

The indices produced by the hash function are approximately uniform over the set of real weights. This produces a weight matrix in which weights are randomly tied, with each unique weight occurring on average the same number of times. [28] demonstrate that the cost of accessing these weights is negligible at test time. In our experiments, we do not use a hash function, instead sampling the indices once when the layer is initialised and storing them.

The number of parameters to be optimised here is the number of *real* weights $\mathbf{w}$, which can be set to be 1 or greater, up to the number of elements in the virtual weight matrix. However, as the number of real weights is increased the probability we may store a weight that is never used in the virtual weight matrix increases. If N_r is the number of real weights and N_v is the number of virtual weights, then the expected number of weights that will be excluded will be $N_r(1 - 1/N_r)^{N_v}$. Defining N_r in terms of N_v using a compression ratio $c = \frac{N_r}{N_v}$, we can investigate what happens to the ratio excluded, e, as c changes:

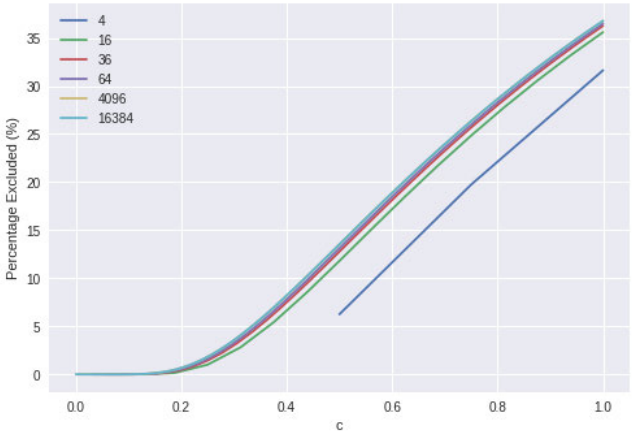

FIGURE 8. The effect on percentage of weights excluded depending on compression ration c, tested for different values of N_v, the number of elements in the virtual weight matrix, indicated in the legend. At the compression levels we are interested in—20% of the original number of weights—we can see that the number of weights excluded is low.

$$e = \left(1 - \frac{1}{cN_v}\right)^{N_v}. \tag{10}$$

Taking the limit in the case of large N_v, we can see this limit has the functional form of the limit definition of an exponential, $\exp(x) = \lim_{N\to\inf}\left(1 + \frac{x}{N}\right)$:

$$\eta = \lim_{N_v\to\inf}\left(1 - \frac{1}{cN_v}\right)^{N_v} = \exp\left(-\frac{1}{c}\right) \tag{11}$$

As shown in Figure 8, this limit argument holds true for the values of N_v we are interested in, and the proportion of weights excluded as the compression ratio grows can be significant. In our experiments we do not address these wasted parameters, despite performing experiments with compression ratios in regions where 10-20% of our parameters are being excluded. It would also be possible to identify these parameters and choose not to store them, but we do not

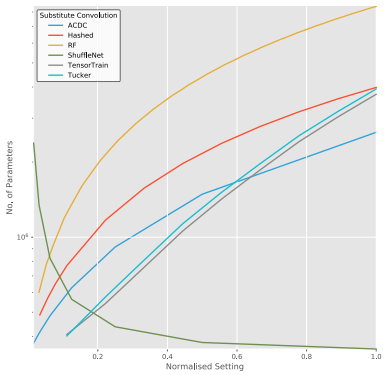

FIGURE 9. The parameter cost of a WRN-28-10 after substitution by the methods listed in the legend, varying the tunable parameter of each over a normalised range. We design experiments over a parameter count range such that all methods illustrated will have support, which here is limited by the maximum size of the Linear ShuffleNet and the minimum size of the RF substitution.

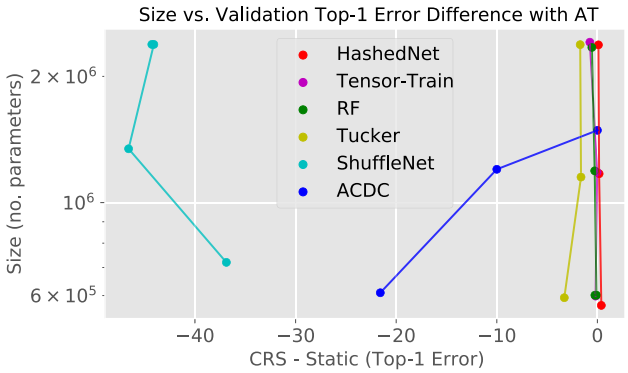

FIGURE 10. The difference in top-1 error on the validation set of CIFAR-10, when training with and without CRS weight decay, over all the substitution methods considered for WRN-28-10. For all methods apart from HashedNet, this form of weight decay scaling is beneficial; it results in a lower top-1 validation error.

investigate this. The reason being that we find the HashedNet substitution effective at high compression levels, such as below $c = 0.1$, and in this region a negligible number of weights will be excluded.

APPENDIX B
CHOSEN PARAMETER BUDGETS

After normalising the tuning of all layers between 0 and 1, we can plot number of parameters used by each substitution as shown in Figure 9. The upper limit and lower limits were chosen where all methods have support. For example, we can see in Figure 9 we can see that the upper limit is defined by the Linear ShuffleNet, while the lower limit is defined by RF. We chose the midpoint by linear interpolation in log parameter count.

APPENDIX C
CRS WEIGHT DECAY ABLATION

To justify CRS weight decay, we ran an ablation experiment, repeating the experiments on CIFAR-10 with WRN-28-10, but disabling CRS weight decay. In Figure 10 these results are illustrated. For almost all methods we see that there is a

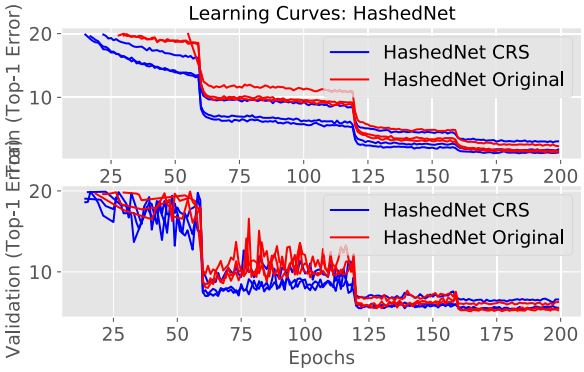

FIGURE 11. Learning curves for HashedNet substitution experiments, with/without CRS weight decay. When CRS weight decay is enabled the top-1 error is lower, on train and test, at every epoch until the final part of the learning rate schedule.

clear benefit. ShuffleNet simply fails to converge without it. However, for HashedNet we see that it is slightly detrimental.

To investigate why this happens, Figure 11 illustrates the learning curves—top-1 error plots against current training epoch—of these HashedNet substitute networks. The CRS weight decay stabilises training as we would hope, and the top-1 validation error is lower with it enabled *until the final stage* of the learning rate schedule. At this stage we can see the training top-1 error decreases faster when CRS weight decay is enabled. This overfitting is enough to cause a slight increase in top-1 error.

ACKNOWLEDGMENT

The authors thank Rafael Ballester for correspondence on tensor-train practicalities, and Joseph Mellor for productive discussions on the CRS weight decay scheme and HashedNet. The opinions expressed and arguments employed herein do not necessarily reflect the official views of these funding bodies.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[2] P. Warden. (2018). *Why the Future of Machine Learning is Tiny*. [Online]. Available: https://tinyurl.com/yculf7wu
[3] V. Radu, C. Tong, S. Bhattacharya, N. D. Lane, C. Mascolo, M. K. Marina, and F. Kawsar, "Multimodal deep learning for activity and context recognition," in *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, 2018, vol. 1, no. 4, p. 157.
[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015. [Online]. Available: https://dblp.org/rec/journals/corr/SimonyanZ14a
[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[6] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 2818–2826.
[7] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 4700–4708.
[8] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.
[9] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7132–7141.

[10] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1284–1293.
[11] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, 1989, pp. 598–605.
[12] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[13] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=SJGCiw5gl
[14] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, "Faster gaze prediction with dense networks and Fisher pruning," 2018, *arXiv:1801.05787*. [Online]. Available: http://arxiv.org/abs/1801.05787
[15] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2019. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7
[16] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," 2016, *arXiv:1602.02830*. [Online]. Available: http://arxiv.org/abs/1602.02830
[17] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=S1_pAu9xl
[18] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.
[19] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=HyQJ-mclg
[20] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=rJ8uNptgl
[21] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," 2014, *arXiv:1412.6806*. [Online]. Available: http://arxiv.org/abs/1412.6806
[22] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg
[23] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX
[24] L. J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2654–2662.
[25] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*. [Online]. Available: http://arxiv.org/abs/1503.02531
[26] L. Sifre, "Rigid-motion scattering for image classification," M.S. thesis, Dept. Comput. Sci., École Polytechnique, Palaiseau, France, 2014.
[27] E. J. Crowley, G. Gray, and A. Storkey, "Moonshine: Distilling with cheap convolutions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2888–2898.
[28] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.
[29] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 442–450.
[30] M. Moczulski, M. Denil, J. Appleyard, and N. de Freitas, "ACDC: A structured efficient linear layer," in *Proc. Int. Conf. Learn. Represent.*, 2016. [Online]. Available: https://openreview.net/forum?id=4ONXivIYX7w
[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1–9.
[32] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "CuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*. [Online]. Available: http://arxiv.org/abs/1410.0759
[33] P. Gibson, J. Cano, J. Turner, E. J. Crowley, M. O'Boyle, and A. Storkey, "Optimizing grouped convolutions on edge devices," in *Proc. IEEE 31st Int. Conf. Appl.-Specific Syst., Archit. Process. (ASAP)*, Jul. 2020, pp. 189–196.
[34] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.
[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861
[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[37] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: http://arxiv.org/abs/1602.07360
[38] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi, "Deep roots: Improving CNN efficiency with hierarchical filter groups," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1231–1240.
[39] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1492–1500.
[40] M. Wang, B. Liu, and H. Foroosh, "Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial 'Bottleneck' structure," 2016, *arXiv:1608.04337*. [Online]. Available: http://arxiv.org/abs/1608.04337
[41] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.
[42] G. Huang, S. Liu, L. V. D. Maaten, and K. Q. Weinberger, "CondenseNet: An efficient DenseNet using learned group convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2752–2761.
[43] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," in *Proc. Int. Conf. Learn. Represent.*, 2015. [Online]. Available: https://dblp.org/rec/journals/corr/JinDC14
[44] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4013–4021.
[45] Z. Yang, M. Moczulski, M. Denil, N. De Freitas, L. Song, and Z. Wang, "Deep fried convnets," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1476–1483.
[46] M. Bojarski, A. Choromanska, K. Choromanski, F. Fagan, C. Gouy-Pailler, A. Morvan, N. Sakr, T. Sarlos, and J. Atif, "Structured adaptive and random spinners for fast machine learning computations," in *Proc. Artif. Intell. Statist.*, 2017, pp. 1020–1029.
[47] K. A. Vahid, A. Prabhu, A. Farhadi, and M. Rastegari, "Butterfly transform: An efficient FFT based neural architecture design," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 12024–12033.
[48] E. Parisotto, L. J. Ba, and R. Salakhutdinov, "Actor-Mimic: Deep multitask and transfer reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016. [Online]. Available: https://openreview.net/forum?id=4zXscyOrI_m
[49] A. Korattikara, V. Rathod, K. Murphy, and M. Welling, "Bayesian dark knowledge," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3438–3446.
[50] D. Lopez-Paz, L. Bottou, B. Schölkopf, and V. Vapnik, "Unifying distillation and privileged information," 2015, *arXiv:1511.03643*. [Online]. Available: http://arxiv.org/abs/1511.03643
[51] V. Vapnik and R. Izmailov, "Learning using privileged information: Similarity control and knowledge transfer," *J. Mach. Learn. Res.*, vol. 16, pp. 2023–2049, Jan. 2015.
[52] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4320–4328.
[53] T. Chen, I. Goodfellow, and J. Shlens, "Net2Net: Accelerating learning via knowledge transfer," 2015, *arXiv:1511.05641*. [Online]. Available: http://arxiv.org/abs/1511.05641
[54] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," in *Proc. Int. Conf. Learn. Represent.*, 2015. [Online]. Available: https://dblp.org/rec/journals/corr/RomeroBKCGB14
[55] G. Urban, K. J. Geras, S. E. Kahou, O. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose, and M. Richardson, "Do deep convolutional nets really need to be deep and convolutional?" in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=r10FA8Kxg
[56] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=Sks9_ajex

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.
[58] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf.*, 2016, pp. 87.1–87.12.
[59] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[60] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, Jan. 2011.
[61] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*. [Online]. Available: http://arxiv.org/abs/1511.06530
[62] R. Zhao, Y. Hu, J. Dotzel, C. D. Sa, and Z. Zhang, "Building efficient deep neural networks with unitary group convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11303–11312.
[63] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
[64] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Proc. Int. Conf. Learn. Represent.*, 2016. [Online]. Available: https://dblp.org/rec/journals/corr/YuK15
[65] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.
[66] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[67] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 646–661.
[68] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
[69] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
[70] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 19–35.
[71] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.
[72] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017, *arXiv:1708.04552*. [Online]. Available: http://arxiv.org/abs/1708.04552
[73] J. Turner, E. J. Crowley, M. O'Boyle, A. Storkey, and G. Gray, "Block-Swap: Fisher-guided block substitution for network compression on a budget," in *Proc. Int. Conf. Learn. Represent.*, 2020. [Online]. Available: https://openreview.net/forum?id=SklkDkSFPB

ELLIOT J. CROWLEY received the M.Eng. and D.Phil. degrees in engineering science from the University of Oxford, in 2012 and 2016, respectively.

From 2016 to 2020, he was a Postdoctoral Researcher with the School of Informatics, The University of Edinburgh, where he primarily worked on making neural networks more efficient. Since 2020, he has been a Lecturer (equivalent to an Assistant Professor) in machine learning and computer vision with the School of Engineering, The University of Edinburgh. His thesis was on object recognition and detection in art using machine learning.

GAVIN GRAY was born in Edinburgh, U.K. He received the M.Eng. degree in electrical engineering and electronics, the master's degree by research in neuroinformatics, and the Ph.D. degree in neuroinformatics from The University of Edinburgh, in 2013, 2014, and 2019, respectively.

He currently works as a Postdoctoral Fellow with the Vector Institute, University of Toronto. His previous publications include *Moonshine: Distilling with Cheap Convolutions* (NeurIPS, 2018) and *Resource-Efficient Feature Gathering at Test Time* (Reliable Machine Learning in the Wild, NeurIPS, 2016). His present work, with Sageev Oore, focuses on rythmic auditory stimulation for Parkinson's disease. His Ph.D. thesis is focused on efficiency in machine learning.

JACK TURNER received the bachelor's degree in computer science from the University of Birmingham. He is currently pursuing the Ph.D. degree with the School of Informatics, The University of Edinburgh.

He is supervised by Prof. Michael O'Boyle as a Student with the EPSRC Centre for Doctoral Training in Pervasive Parallelism, where he studies resource efficiency in deep neural networks and related compiler technology.

AMOS STORKEY received the M.A. degree in mathematics from the University of Cambridge, and the M.Sc. and Ph.D. degrees in machine learning from Imperial College, London.

He moved to The University of Edinburgh after his Ph.D. He is currently a Professor of machine learning and AI with the School of Informatics, The University of Edinburgh, where he leads a research team focused on deep neural networks, Bayesian and probabilistic models, transactional machine learning, and efficient inference. He balances work on fundamental methods in machine learning and application across a spectrum of research areas from medical imaging, games, and music amongst others. He is also the Director of the EPSRC Centre for Doctoral Training in Data Science. He has been a Programme Chair for the AI and Statistics Conference. He was a Founder of the Edinburgh Deep Learning Workshop.